RPG System– Design Document

By: Sarthak Saxena

# Brief Introduction

The RPG System will be designed to help developers create their Role-Playing Game with more flexibility. The system consists of a complex system that helps a developer with character creation, weapons creation, enemy creations, attack phase, skills creations and items.

There are a few preset scripts to be attached to the relevant game objects, such as the items, actions, and character script. The developer can access the main scriptable object, which allows the developer to interact with a custom inspector. The custom inspector will be controlled through an advanced algorithm, detailed further below.

The RPG System Controller allows the developer to create a character, weapons, items and actions based on their selected options. The developer may choose to make a RPG style character which holds basic primary stats. As well as directly attach actions, weapons/items which target allies, targets, or self (“player”). The system allows the developer to create items/weapons to effect the stats of the enemies or “player”. Furthermore, the developer may attach actions to the “player/enemy” array and identify what type of effect it will have on the opposing or self-stats straight from the custom inspector.

The system has its own event handlers such as the action/fight phase in which the calculation of the damage, defense and health stat will be prioritized. There will be an effect phase in which all de-buffs/buffs will output the result, and will decrement from their total life time. The system makes it easier for developers to add in similar/same stats on weapons, items, buffs, de-buffs by only including the given stats on the “player” or “enemy” target to be modified or added.   
  
The developer can create any RPG game using the system, as the system is designed to be modular. The system if used correctly can be used to implement any game mechanic as long as the developer chooses to.

The system will be a Unity package which will have all the items required for the system to work. Unity package will include a few scripts for the main system script to access. It will include scripts such as a preset items script, entity script and the main RPG script.

**Entity Preset Script**  
The entity script will include an enum of character types such as “enemy”, “player” and “ally”. It will include class member variables such as arrays of Class Stat, Class Item, and Class Action.   
  
The enums used are composed of the following:

|  |  |
| --- | --- |
| ENUM | Variables |
| CharacterType | * Enemy * Player * Ally |
| StatType | * characterHitPoints * chracterEnergy * characterDamage * characterDefense * characterAccuracy * characterDodge |

The class of stat is composed of the following:

|  |  |
| --- | --- |
| Class | Member Variables |
| Stat | * StatType typeOfStat * float statValue * bool statusEffect |

These stats are composed of StatType which is an enum that refers to the type of stat it is. The developer may create a health core stat, by selecting the type characterHitPoints. This will get saved into the Stat object, so when the RPG System Controller references any method which refers to an entity’s health, it can find it. The statusEffect Boolean can be ticked by the developer so the stat will act as a status effect rather than a permanent value.

The character class preset script will contain the following:

|  |  |
| --- | --- |
| Class | Member Variables |
| Character | * CharacterType type * Stat [] characterStats * Item [] characterItems * Action [] characterActions * Stats[] characterStatusEffects |

The developer can attach this script onto their “character” GameObject and then attach the “character” to the main RPG system inspector. The inspector will then call the main RPG system script which will allow the developer to add any “stats”, “items”, “actions” and “status effects”.

**Item Preset Script**The preset script will have the item class as outlined:

|  |  |
| --- | --- |
| Class | Member Variables |
| Item | Stat[] affectedStats |

Once the developer attaches the script onto a “item” game object in their Unity inspector, the item will be ready for modification. The developer can attach this “item” to the main script in the custom inspector from which the stat buffs/de-buffs can be added, of the given character types.  
  
**Action Preset Script**The Action preset script will contain the following class and variables:

|  |  |
| --- | --- |
| Class | Member Variables |
| Action | * float cost * GameObject Target * Stat[] affectedStats |

# 3rd Party Libraries

Since the engine of choice is Unity and the package requires Unity, the 3rd party library in use is Unity.

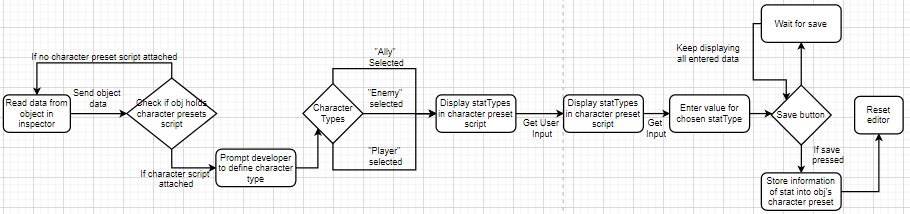
# Advanced Algorithms

## RPG Script

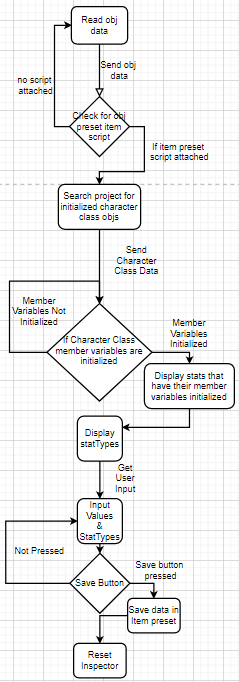
The single script holds different sections for each RPG mechanism.   
These sections include:

### Character Creator

The developer will first attach the preset character script onto their “character”, and then attach the “character” onto the main RPG script custom inspector.   
Now the following algorithm takes place:

This algorithm checks if the “character” holds a character script so that the main RPG script may access its member variables. First the editor will prompt the developer to define the character’s type, which could be ‘player’, ‘enemy’ or ‘ally’. The editor will then display an option to create core stats for the character, which will be used in all equations of the game. The developer will be shown multiple options of types such as character health, damage, defense, energy, etc. from these the developer can choose the stat type. Once the stat type has been chosen the value can be set as a default starting point from the inspector. After the developer has finished making the stat they may press ‘save’ which reverts the inspector to the beginning. Additionally, the ‘save’ will store the character stats in the character preset script attached to the character.

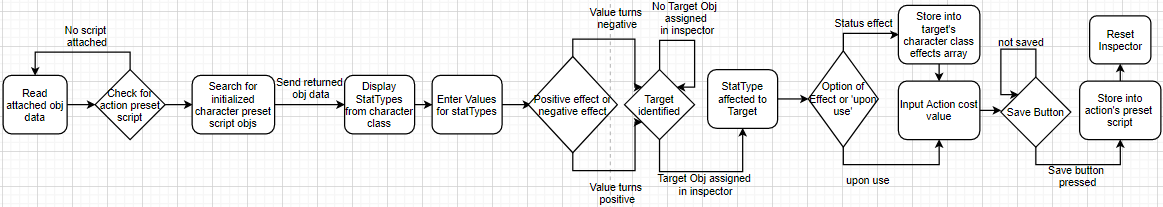
### Item/Weapon Creation

The following section will allow the developer to create an item that may act as a weapon or an accessory. The developer needs to first attach the item preset script onto the “item” game object in Unity. After the “item” holds the item preset script it is ready to be assigned to the inspector. The following algorithm allows developers to create an item with StatTypes (“characterDamage”) that are already in the game rather than double checking the correct StatTypes to use. This is done by searching for a “character” that holds the character script with initialized values. Once this search returns an initialized character, it will loop through the stats owned by the character and display the statTypes of those stats. These will be displayed through the Unity inspector, from which the developer will be given options of statTypes consisting of the ones returned from the search done previously. After, the developer has added in a statType and value, they may click ‘save’ which will store that information onto the item preset script attached to the item. Additionally, this will reset the item section of the inspector so that the developer may attach another “item”.

### Actions

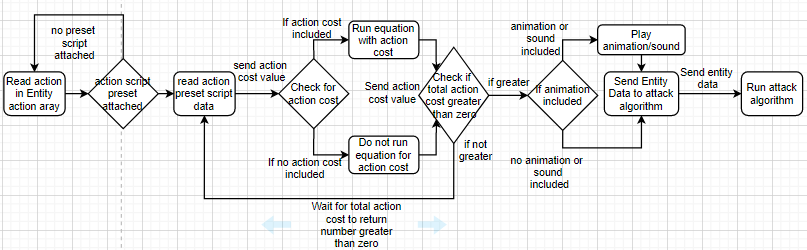
This section contains any actions that the developer wants to add to the character. The “characters” hold an array of “actions” that can be created, or attached directly to the “character”. The developer can attach the preset action script to the “action” game object that they wish to attach to the “character”. Once the developer has attached this they may attach it to the main RPG script to create an action. The following algorithm, will read the values of the object attached to check for the preset “action” script. Afterwards, the script will read previously made “characters” and fill all options of statTypes according to the initialized statTypes.

Furthermore, once the stat is created with the statType, they will be prompted to enter a value for it to act as a positive effect or negative effect. Additionally, the developer may then add in a target for the action. This may be done by the developer during run-time or through editor by other means.

The developer can then add whether if it will be an effect over time or just an ‘upon used’ effect. If the effect option is chosen, the stat will act as a status effect that will be stored into the target’s characterStatusEffects array. At the end, they will be prompted of the cost of the action to be used. Eg. 10 mana required for a fireball. They will finish then by clicking ‘save’ which will save the settings to the action preset script.

### Attacking

In this section the following algorithm will run a simulated attack phase which can be called by the developer that outputs the character stats afterwards. Once called, the algorithm will read the two entities referenced for the attack by the developer. It will check for the action cost, to see if an action can be played by running the following equation:  
**character energy = character energy – action cost.**  
If the character’s energy is above zero then the action is valid and can be performed, so the function continues. Once the action is valid, loop through each entities’ items and add to the correct statType.



First looping though each stat on the items, if the item has statType characterDamage with the value of 4, the value will be added to the Entity’s characterDamage stat. This will continue for each stat until the end of the items list. After the strength and defense values have been accumulated from looping through the entity’s item it will check if the entity holds an accuracy or dodge stat. If this returns true, the following equation will be used:

To scale the damage to work in accordance to the max health an entity can have, it has been placed at a ‘sweet spot’ of 10. This allows the damage to stay low, so the value doesn’t reach double digits too quick, but if the developer wishes to add more damage it still raises consistently. The developer may change this value if they have higher stat values for characterDamage, such as in the triple digits or more if they wish to.

The buffs and debuffs calculated will be done through a small equation in the beginning:  
Effect is the buff or de-buff found as a float value, which will be added to a total number. **buffs/de-buffs calculated = buffs/de-buffs calculated + Effect**  
The equation will be performed once a buff or de-buff is found.

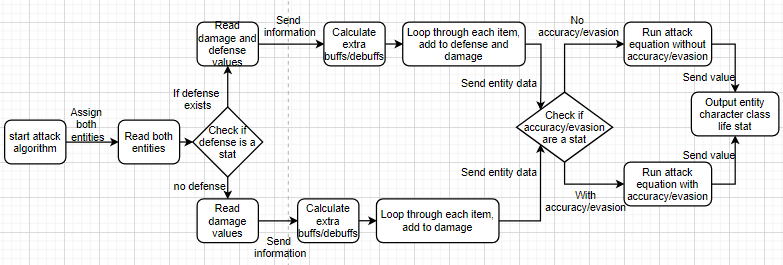
**Total damage done = characterDamage (10/ (10 + characterDefense)) + buffs/de-buffs calculated%  
Total damage done = Total damage done \* characterDodge multiplier%**

Otherwise this will be used if the defense and damage statTypes are assigned:  
**Total damage done = characterDamage (10/ (10 + characterDefense)) + buffs/de-buffs calculated%**

Or this will be used if the defense variable is not assigned but the damage statType is assigned:

**Total damage done = characterDamage + buffs/de-buffs calculated%**

Additionally, the developer may use the outputted values to remove from the health statTypes from the assigned entities.



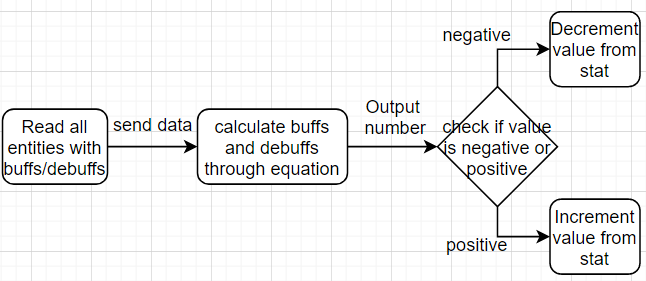
### Buffs/De-buffs

The script will run this algorithm once an ‘item’ has a buff or de-buff, as well as if an ‘action’ was called. The algorithm goes through all entities which contain a buff or de-buff. From there, it will run its equation to calculate if the output of the buffs/de-buffs calculated is negative or positive. If the number returned is positive then the positive number will be added to the stat each frame, or vice versa.

**Current Related Stat Value = Total Stat Buff + Related Character Stat**

As an example, a potion gives the character a 10% percent boost to all stats. On the other hand, the player has a de-buff of ‘bleed’ reducing all stats by 5%. Therefore, the potion will only boost up to 5%.  
Eg. Total stat value = stat + stat, therefore

Total stat value = 10 + (-5) = 5



# Modular

The system is modular as it allows the developers to create characters, items, buffs to be placed in their own RPG style game. The system is packaged with preset item scripts, character scripts and actions scripts which allows any object created by the developer to be turned into an ‘item’, ‘character’ or ‘action’.   
  
System will allow the developer to add in multiple skills, and change values of skills and stats flexibly through the unity editor. This allows the system to be used for each developer for each game as it allows the values to change per game. Values such as accuracy, evasiveness that effect the calculation of damage are added for developers who wish to add in such mechanics to their game, but are not mandatory.

The system allows each developer to build their own type of items which may consist of different statTypes and values. Since this system uses preset scripts to store data, this can be used in any Unity project as long as the main script is added to the scene. The script will refer to the preset scripts when using its calculation allowing any object created by the script to be moved to any project.

# How to Integrate into Project

To use the following package simply un-package the system folder. There will be a read-me file that outlines how to set each preset script onto the objects in Unity to be used for the RPG system. The main script needs to be placed onto any game object in Unity, so that the custom inspector that handles everything can be initialized. The preset scripts will be clearly outlined in their folders, with an appropriate name depicting what they are. The developer may then either drag and drop the preset scripts onto any object, and then drag that object onto the main script editor.